

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

183

The Munich Project CIP

Volume I: The Wide Spectrum Language CIP-L

By the CIP Language Group:

F.L. Bauer, R. Berghammer, M. Broy, W. Dosch,
F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse,
B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl,
H. Partsch, P. Pepper, K. Samelson (†), M. Wirsing
and H. Wössner



Springer-Verlag
Berlin Heidelberg New York Tokyo

Editorial Board

D. Barstow W. Brauer P. Brinch Hansen D. Gries D. Luckham
C. Moler A. Pnueli G. Seegmüller J. Stoer N. Wirth

Authors

F.L. Bauer B. Möller
R. Berghammer H. Partsch
W. Dosch P. Pepper
R. Gnatz K. Samelson(†)
E. Hangel H. Wössner
Institut für Informatik der TU München
Postfach 20 24 20, 8000 München 2

M. Broy
F. Nickl
M. Wirsing
Fakultät für Mathematik und Informatik, Universität Passau
Postfach 2540, D-8390 Passau

F. Geiselbrechtinger
Department of Computer Science, University College Dublin
Belfield, Dublin 4, Ireland

W. Hesse
Softlab GmbH
Arabellastr. 13, D-8000 München 81

B. Krieg-Brückner
Fachbereich 3, Informatik, Universität Bremen
Postfach 33 04 40, D-2800 Bremen 33

A. Laut
PCS GmbH, Periphere Computer-Systeme
Pfälzer-Wald-Str. 36, D-8000 München 90

T. Matzner
sd&m GmbH, Software Design & Management
Führichstr. 70, D-8000 München 80

CR Subject Classification (1982): D.1.0, D.2.1, D.2.4, D.3.1, D.3.3, F.3.1,
F.3.2, F.3.3

ISBN 3-540-15187-7 Springer-Verlag Berlin Heidelberg New York Tokyo
ISBN 0-387-15187-7 Springer-Verlag New York Heidelberg Berlin Tokyo

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to "Verwertungsgesellschaft Wort", Munich.

© by Springer-Verlag Berlin Heidelberg 1985
Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

PREFACE

This book is the first of two volumes that present the main results having emerged from the project CIP - Computer-Aided, Intuition-Guided Programming - at the Technical University of Munich. The central theme of this project is program development by transformation, a methodology which is felt to become more and more important.

Whereas Volume II will contain the description, formal specification, and transformational development of a system, CIP-S, that is to assist a programmer in this methodology, the present volume gives the description and formal definition of a program development language CIP-L designed particularly for use in transformational development of programs from formal specifications. Many sources have influenced the development of this language over the past eight years, and we feel that it has now matured and consolidated to a degree that its study will be profitable to others.

Three aspects of this language appear to be of special interest:

First, the language is a *coherent wide spectrum language*. This means that it comprises a number of expressive levels ranging from predicative and algebraic specifications over applicative and procedural constructs (including parallelism) down to a machine-oriented level using jumps and pointers. However, these levels do not just form a loose collection of features; rather they are closely linked by formal transformation rules that relate the various constructs to give a coherent semantics to the entire language. Besides, these rules also give a guideline for the overall structure of the transformational development process. Since CIP-L comprises most of the essential concepts of today's programming languages in more or less similar form, the rules also provide some insight into the general structure of programming languages and programming.

The second major aspect of interest in this book is the definition of the language by the new method of *transformational semantics*. In this approach, a kernel language is distinguished as that part of the language which contains the essential semantic concepts. All other language constructs are then mapped into this kernel by formal transformation rules that allow reducing every program to an equivalent one of the kernel language. It seems remarkable that the set of rules needed for the particular language CIP-L is quite small and easy to survey; in most cases just one rule per additional language construct suffices. Thus, a transformational definition of a well-designed language is as concise as, say, a denotational one; moreover, in our opinion, it is much easier to comprehend.

As a third major aspect, CIP-L is an *abstract scheme language*. This means, first, that its formal definition works on the abstract syntax of the language. As a consequence, CIP-L allows various concrete syntactic representations, two of which, viz. an ALGOL-like and a PASCAL-like form, are given in syntax charts in an appendix. Second, CIP-L is a scheme language in that it is independent of any particular set of data structures (except the truth values); rather it comprises elaborate means for defining new data structures.

The book is organized as follows: Part 0 contains a general introduction to transformational programming and to the project CIP in particular. In Part I, the various constructs of the language are introduced informally together with examples of their place in program development. Part II, the heart of the book, then gives a description of the language in a systematic way. However, readability is considered more important than complete formality; in particular, the transformation rules are given in concrete syntax, and a number of self-evident context conditions are omitted. Finally, Part III contains a complete formal definition of the language in the same order of presentation as Part II.

Cross-references within one part are given by section numbers only; references to other parts are made by prefixing the respective section numbers with the (roman) part numbers.

We would like to express our thanks to the Deutsche Forschungsgemeinschaft who has sponsored this research within the Sonderforschungsbereich 49 "Programmiertechnik" during the past nine years. Also, we gratefully acknowledge valuable criticism of the language and of its definition by the lecturers and participants of the Marktoberdorf Summer Schools 1978 and 1981, notably by E.W. Dijkstra, D. Gries, and C.A.R. Hoare, as well as by the members of IFIP Working Group 2.1, notably by H.J. Boom, G. Goos, L.G.L.T. Meertens, S.A. Schuman, and M. Sintzoff. Particular thanks are due to R.S. Bird for pointing out a severe error in an earlier version of the semantic specification of the kernel language. Last, but by no means least, we gratefully acknowledge many helpful remarks by our colleagues C. Delgado Kloos, H. Ehler, F. Erhard, U. Hill-Samelson, A. Horsch, H. Hußmann, W. Meixner, R. Obermeier, H.-O. Riethmayer, G. Schmidt, and R. Steinbrüggen.

Munich, December 1984

The CIP Language Group

TABLE OF CONTENTS

<u>PART 0 : INTRODUCTION TO THE PROJECT CIP</u>	1
0. History of the project CIP	3
1. A wide spectrum language for inferential programming - survey of CIP-L	4
1.1. Overall-structure and concepts of the language CIP-L	5
1.2. Remarks on wide-spectrum languages for non-conventional architectures	7
2. Methodology of program development	8
2.1. The legal process of programming	9
2.2. The economical process of programming	9
2.3. The social process of programming	10
3. The technical process of inferential programming - survey of the program transformation system CIP-S	10
<u>PART I : INFORMAL SURVEY OF THE LANGUAGE</u>	13
1. The expression language for logic and functional programming	15
1.1. Expressions	15
1.1.1. Guarded expressions	15
1.1.2. Function abstraction and application	16
1.1.3. Functions defined as fixpoints (recursion)	18
1.1.4. Higher-order functions	20
1.1.5. Finite choice	21
1.2. Descriptive constructs for specification purposes	22
1.2.1. Description	22
1.2.2. Comprehensive choice	22
1.2.3. Descriptive sets	24
1.2.4. Quantifiers	26
2. The full applicative language	28
2.1. Object declarations	28
2.2. Function declarations	28
2.2.1. Function declarations in the lambda-calculus style	28
2.2.2. Function declarations in the style of ALGOL-like languages	29
2.2.3. Function declarations in the style of algebraic types	30
2.3. Object declarations with assertions	31
2.4. Consecutive declarations	31

VI

3.	The procedural language	33
3.1.	Variables and statements	33
3.2.	Procedures	35
4.	The control-oriented language	37
4.1.	Iteration statements	37
4.2.	Labels and jumps	38
5.	Parallel constructs	40
6.	Algebraic types, modes, and computation structures	41
6.1.	Algebraic types	41
6.2.	Modes	45
6.3.	Computation structures	48
7.	Modules and devices, arrays, pointers	52

PART II : DESCRIPTION OF THE LANGUAGE 53

1.	Algebraic types	55
1.1.	Definition of types	55
1.1.1.	Introduction	55
1.1.2.	The constituents	56
1.1.3.	Primitive types	56
1.1.4.	The signature of a type	57
1.1.5.	The laws of a type	58
1.1.6.	The semantics of a type	60
1.2.	Type schemes	61
1.2.1.	Parameterized type specifications	61
1.2.2.	Instantiations of type schemes	62
1.2.3.	Restrictions on parameters of type schemes	65
1.2.4.	Type schemes without constituents	65
1.3.	Modes	66
1.3.1.	Product	67
1.3.2.	Sum	67
1.3.3.	Recursive modes	70
1.3.4.	Notational extensions	73
1.4.	Standard types	75
	(a) Boolean values	75
	(b) Integral numbers	76
	(c) Finite sets and multisets	77
	(d) Finite mappings	79
	(e) Sequences	80

(f) Trees	82
(g) Pointers and plexuses	83
2. The scheme language	85
2.1. The kernel: an expression language for logic and functional programming	88
2.1.1. Semantic notions	88
2.1.2. Algorithmic constructs	93
(a) Basic identifier	93
(b) Conditional expression	93
(c) Tuple	95
(d) Finite choice	96
(e) Function abstraction	97
(f) Function application	98
(g) Fixpoint	100
Notational extension: function composition	101
2.1.3. Prealgorithmic constructs	103
(a) Universal equality test	103
(b) Quantification	105
(c) Comprehensive choice	106
(d) Description	107
(e) Descriptive set constructs	108
2.1.4. Further notational extensions	111
2.1.4.1. Guarded expression	111
2.1.4.2. Partial application	113
2.1.4.3. Function tupling and function construction	115
2.1.4.4. Assertions	116
(a) Parameter and result restrictions	117
(b) Quantification over restricted domains	117
(c) Restriction of sets and functionalities	118
2.2. The full applicative language: declarations	119
2.2.1. Survey	119
2.2.2. Semantics of the full applicative language	120
(a) Object declaration	120
(b) Function declaration	120
(c) Section	121
2.2.3. Notational extensions	122
(a) Postponed declaration	122
(b) Assertion for an object declaration	122
(c) Function declaration in the style of algebraic types	123
2.3. The procedural language: program variables and procedures	124
2.3.1. Survey	124
2.3.2. Semantics of the procedural language	124
Context conditions and attributes	124
Variables and statements	125

(a) Segment and phrase	125
(b) Declaration of variables	126
(c) Assignment	126
(d) Empty statement	127
(e) Abort statement	127
(f) Block	128
(g) Conditional statement	128
(h) Finite choice	128
Extension: Functions that read global variables	129
Procedures	130
(i) Procedure declaration	130
(j) Procedure call	132
2.3.3. Notational extensions	133
(a) Guarded statement	133
(b) Local assertion	134
(c) Non-initialized program variable	135
2.4. Constructs for parallel programming with shared variables	136
2.4.1. Survey	136
2.4.2. Context conditions and attributes	136
2.4.3. Definitional transformations	137
2.5. The control-oriented language: labels and jumps	139
2.5.1. Survey	139
2.5.2. Semantics of the control-oriented language	139
Context conditions and attributes	139
Definitional transformations	140
2.5.3. Extensions	144
(a) Return jump	144
(b) Loops	144
3. Programs	146
3.1. Computation structures	146
3.1.1. Declaration of computation structures	146
3.1.2. Computation structures as implementations of types	147
3.1.3. Parameterized structures	149
3.2. Extension: Submodes	151
3.3. Modules	154
Local instantiations	154
3.4. Devices	155

<u>PART III : FORMAL DEFINITION OF THE LANGUAGE</u>	157
0. Fundamentals of the description	159
0.1. Language levels as a hierarchy of signatures	159
0.2. Abstract syntax of the language	159
0.3. Context conditions	161
0.4. Semantic specification	161
0.5. Table of identifiers	162
1. Algebraic types	163
1.1. Abstract syntax of types	163
1.2. Attributes and context conditions	164
1.2.1. Constituents	165
(a) Sort	165
(b) Operation	166
(c) Constant	166
1.2.2. Terms	167
(d) Free identifier and constant	167
(e) Application	167
1.2.3. Laws	168
(f) Equation	168
(g) Logical connective	168
(h) Quantification	168
1.2.4. Facets of a type	169
(i) Constituent	169
(j) Law	169
(k) Primitive facet	170
1.3. Semantics of non-parameterized types	170
2. The scheme language	172
2.1. The kernel: an expression language	173
2.1.1. Semantic domains	174
2.1.2. Abstract syntax	176
2.1.3. Specification of the algorithmic constructs	176
(a) Identifier	177
(b) Conditional expression	177
(c) Tuple	178
(d) Finite choice	179
(e) Function abstraction	179
(f) Function application	180
(g) Fixpoint	180
2.1.4. Specification of the prealgorithmic constructs	181
(a) Universal equality test	181
(b) Quantification	182

(c) Comprehensive choice	183
(d) Set comprehension	183
(e) Element relation	184
2.2. The full applicative language: declarations	185
2.2.1. Abstract syntax	185
2.2.2. Specification	185
(a) Object declaration	185
(b) Function declaration	186
(c) Segment	187
2.3. The procedural language: variables and procedures	188
2.3.1. Abstract syntax	188
2.3.2. Specification	188
(a) Segment	189
(b) Declaration of variables	189
(c) Assignment	190
(d) Empty statement	190
(e) Abort statement	191
(f) Block	191
(g) Conditional statement	192
(h) Finite choice	192
(i) Procedure declaration	193
(j) Procedure call	194
2.4. Constructs for parallel programming with shared variables	196
2.4.1. Abstract syntax	196
2.4.2. Specification	196
(a) Elementary statement	199
(b) Parallel block	199
(c) Guarded statement	200
(d) Await-statement	201
(e) Parallel composition	201
(f) Procedure with parallel statements	202
(g) Call of a parallel procedure	204
2.5. The control-oriented language: labels and jumps	207
2.5.1. Abstract syntax	207
2.5.2. Specification	207
(a) Elementary statement	207
(b) Labelled statement	209
(c) Goto statement	209
(d) Conditional statement with goto's	210
(e) Finite choice with goto's	211
(f) Block with goto's	212

3.	Programs	214
3.1.	Abstract syntax	214
3.2.	Attributes and context conditions	214
3.2.1.	Facets of components	215
3.2.2.	Definition of parameterized components	216
3.2.3.	Instantiation of parameterized components; type basing	218
	(a) Component instantiation	218
	(b) System of type instantiations	219
3.2.4.	Programs	219
3.3.	Semantics of programs	220
3.3.1.	Normalization	220
3.3.2.	Semantics of normalized programs	227
	(a) Semantics of types	227
	(b) Semantics of structures	227
REFERENCES		228
APPENDIX I : CONCRETE REPRESENTATION OF ABSTRACT PROGRAMS		235
APPENDIX II : BIBLIOGRAPHY OF THE PROJECT CIP		237
APPENDIX III : SYNTAX DIAGRAMS FOR THE ALGOL-LIKE AND THE PASCAL-LIKE EXTERNAL REPRESENTATIONS		253